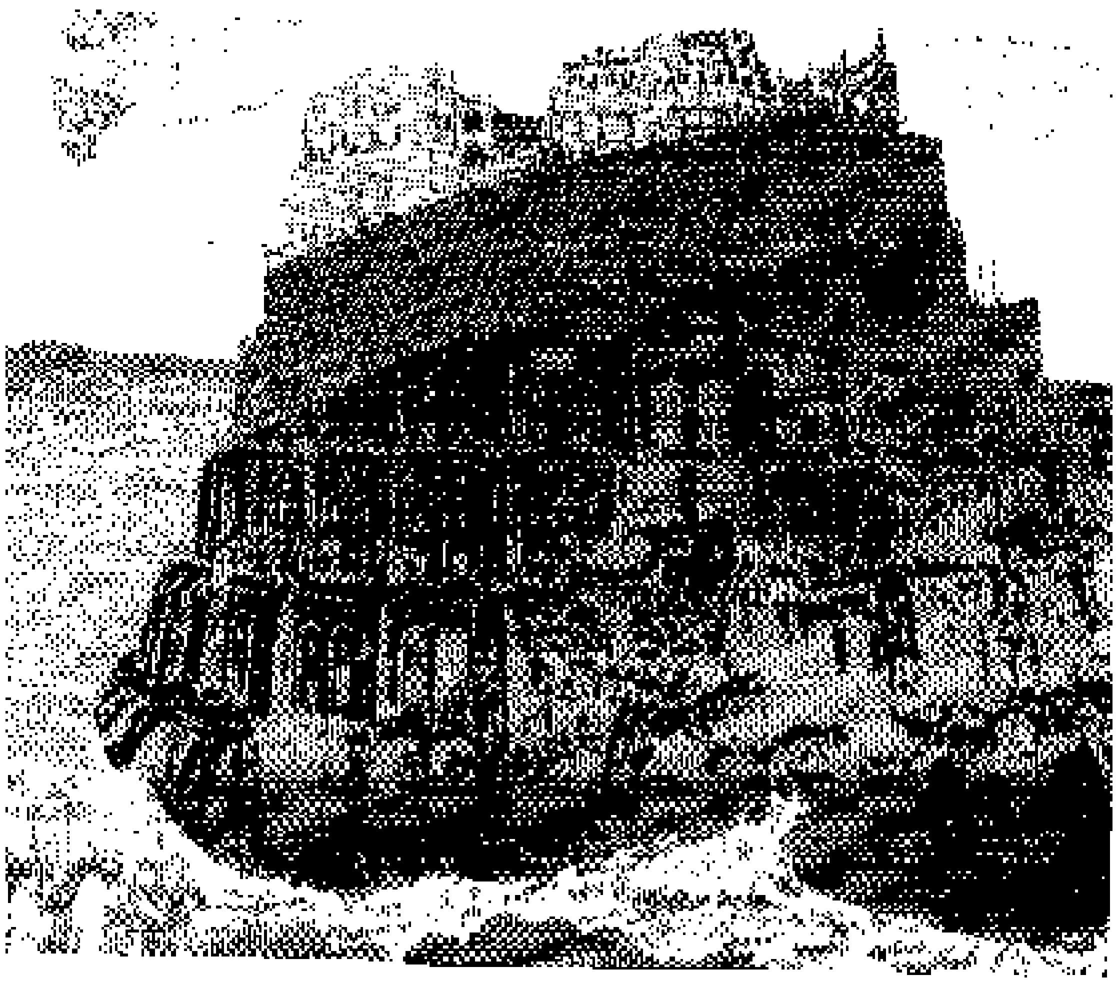
The AI-Workbench BABYLON

A Short Description



GMD
Institute for Applied Information Technology
AI Research Division
PO Box 1316
D-53731 Sankt Augustin
Germany
Juergen.Walther@gmd.de

THE AI-WORKBENCH BABYLON ON THE MACINTOSH

BABYLON is a modular, configurable, hybrid environment for developing expert systems. The following knowledge representation formalisms are provided: objects, rules with forward and backward chaining, Prolog and constraints. BABYLON is implemented and embedded in Macintosh Common Lisp.

At Cebit'89 the book *Die KI-Werkbank BABYLON - eine offene und portable Entwicklungsumgebung für Expertensysteme,* was published by Addison-Wesley. The english version of the book : *The AI Workbench BABYLON* is currently (January 1992) being published by Academic Press. Both books begin with a brief introduction to the foundations of expert systems. Then the knowledge representation formalisms of BABYLON and their interaction are explained. A large, commented example demonstrates how to use the formalisms in a real application. A language extension for component descriptions and diagnosis is presented. Next, the object-oriented implementation is explained so that a systems programmer can adapt BABYLON to his/her special needs.

System requirements: 4MByte main storage, Macintosh Common Lisp 2.0.1 (MCL 2.0.1).

The AI-workbench BABYLON

BABYLON is a hybrid tool system for implementing and operating expert systems. It provides the knowledge engineer with different integrated knowledge representation formalisms and different user interfaces.

BABYLON is configurable:

Interpreters and user interfaces are available in different versions, configurable in any combination to obtain problem-specific tools.

BABYLON is an open tool system:

If the user develops his own knowledge representation formalisms or changes existing ones, he can easily integrate them in BABYLON. The BABYLON architecture is open for such extensions.

BABYLON is portable:

We modularized BABYLON in a strictly functional way, factored out the I/O-operations, and extended the implementation language Common Lisp by our own, small, efficient and portable object system. Thus BABYLON became an easily portable system, as is demonstrated by more than a dozen successful portations to seven different Lisp systems.

Target group:

Version 2.3 of BABYLON is primarily designed as a system for research and development, teaching and training.

Knowledge representation in BABYLON:

BABYLON offers the following languages for knowledge representation:

Prolog

objects

production rules

onstraints

The BABYLON paradigm:

BABYLON provides an independent specialist for each formalism, i.e. in metaphoric terms:

a logician accustomed to proceed analytically, to decompose goals into subgoals and problems into subproblems;

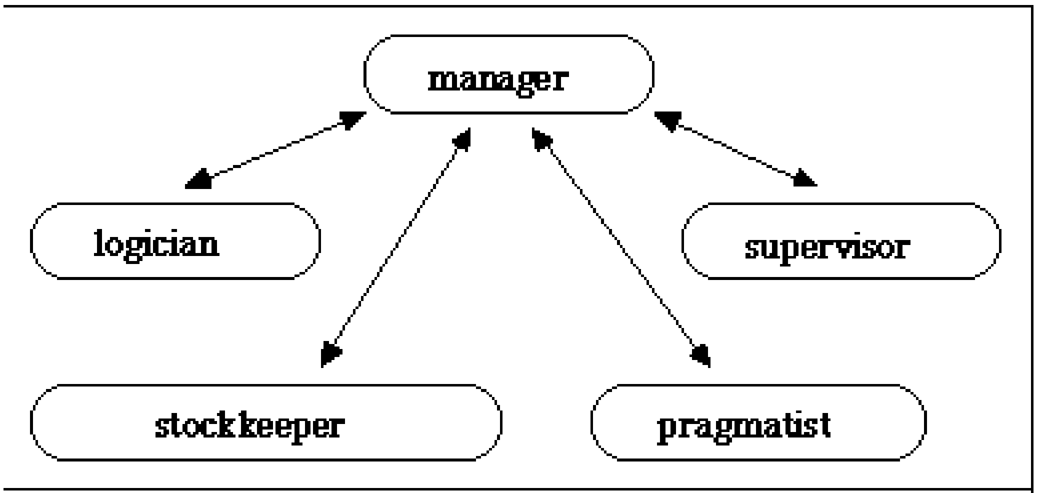
a stockkeeper possessing information on well-known things (objects), managing this information and thus rousing expectations. Of course, the stockkeeper can extend the stock, update old information etc.;

a pragmatist living fully in present-day reality, being concentrated on the concrete and r ecommending actions to be performed in accordance with the specific situation;

a supervisor controlling the performance of actions, asking for future consequences and tracing back the reasons for current situations.

The interaction of the specialists follows the principle of distributed problem solving. There is an other specialist for coordination:

the manager receiving tasks to be accomplished, selecting the suitable specialist, delegating the task and securing the delivery of the result to the correct place.



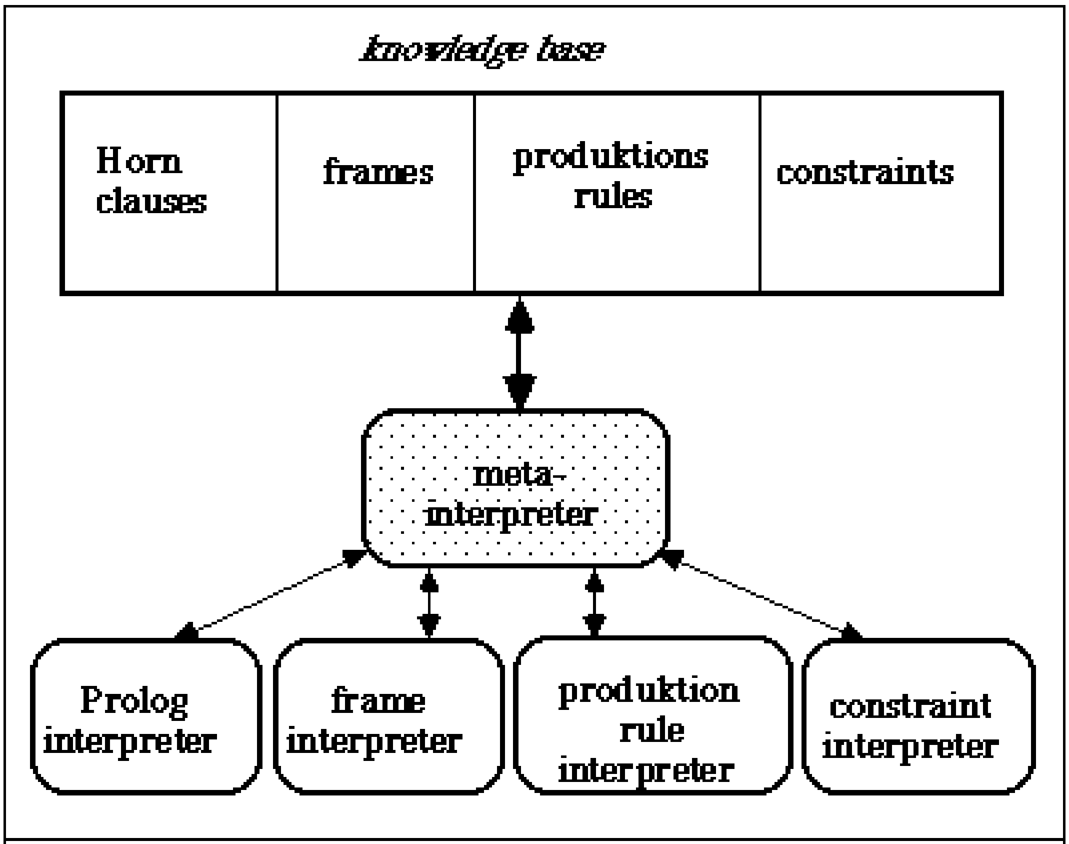
organizational structure of BABYLON

The very point of this division of labor is that the specialists do not know each other. This makes it very easy to omit specialist groups or to compose new ones.

The architectural concept

The BABYLON architecture is a processing model for the organizational structure described above. The specialists are software modules responsible for the interpretation of different formalisms: as a stockkeeper, the frame interpreter interprets the object constructs (frames), the rule interpreter interprets the production rules as a pragmatist, the Prolog interpreter interprets the

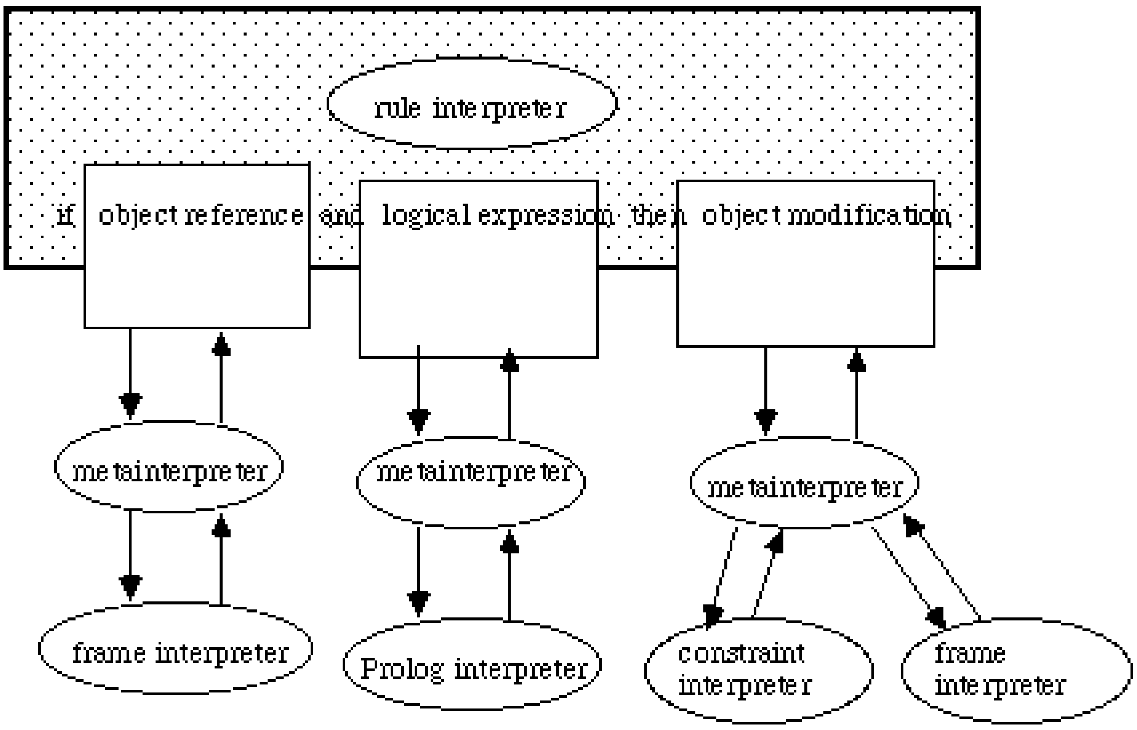
Horn clauses as a logician and the constraint interpreter the constraints as a supervisor. The metainterpreter coordinates the four language interpreters and manages the references made to expressions of other formalisms within a formalism.



BABYLON architecture

The arrows show the communication paths (data and control flow). The arrow connecting the metainterpreter with the knowledge base is also to be regarded as a data and control flow channel. It is used for transmitting the contents of the knowledge base parts to the responsible interpreters. Such a part which is not represented in the figure is the so-called instruction part where the control statements for the metainterpreteritself are filed.

The integration is achieved by the fact that, for example, the left-hand side of production rules may consist of logical expressions or references to objects. The interaction is realized as follows. If the rule interpreter wants to apply such a rule, it activates the metainterpreter by sending it a condition from the left-hand side of the rule, and requests the interpretation of the condition. The metainterpreter then identifies the type of the condition. If it is a reference to an object state, it forwards it to the frame interpreter which can find out whether the object concerned is in the respective state or not. Via the metainterpreter, the reply is then returned to the rule interpreter thus enabling it to continue its work. If the first reply is positive and if the left-hand side is a conjunction of conditions, the remaining conditions are handled in the same way.



However, if a condition on the left-hand side is a predicate expression, the metainterpreter activates the Prolog interpreter. It checks the expression by searching for Horn clauses allowing its proof. The reply of the Prolog interpreter is then returned by the metainterpreter to the rule interpreter as described above. If, eventually, the evaluation of the left-hand side of the rule terminates positively, the rule interpreter begins with the evaluation of the actions of the rule in question. For this purpose, it proceeds analogously to the evaluation of the conditions, i.e. the actions are forwarded to the metainterpreter. Actions may, for example, modify object states. Before effecting the modification, the metainterpreter activates the constraint interpreter. The latter checks whether the modification of the objects is consistent with possibly available constraints. If there are no conflicts from the viewpoint of the constraints, the frame interpreter is requested via the metainterpreter to modify the state of the object in question in accordance with the action.

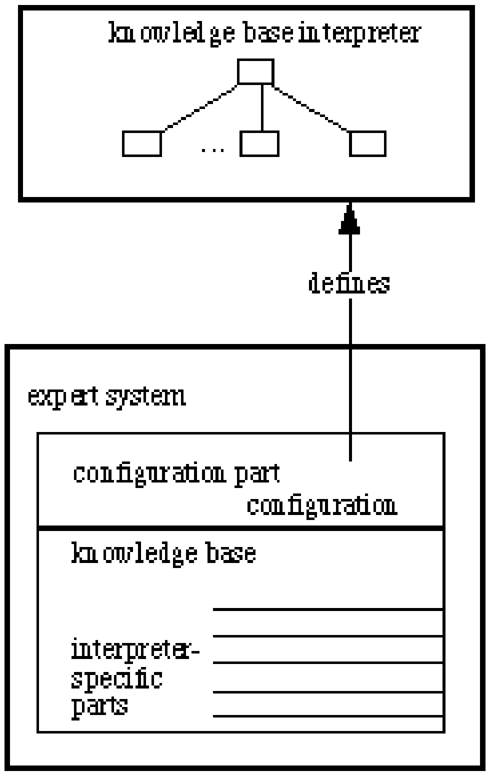
Architectural characteristics:

The basic idea, common to all hybrid systems is that the various integrated formalisms are not provided alternatively, but in a complementary way. The architectural concept distinguishes between several processing levels. At the basic level, which is split into several modules according to the principle of distributing competence and tasks, problem solving processes operate that are coordinated at a higher level (the metalevel). This horizontal and vertical distribution of functionality provides various advantages of openness from the point of view of system technology: openness in depth since the components of the lower level can in principle be put onto any software or hardware basis; openness in breadth since the various basic components can be developed separately and can thus be exchanged or added at any time; openness in height since additional components can be realized by bootstrapping.

BABYLON expert systems:

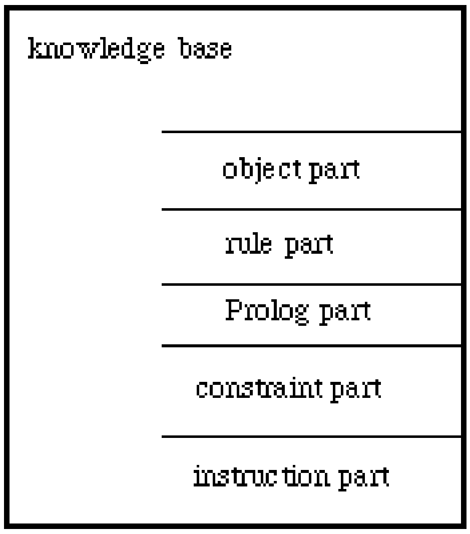
A BABYLON expert system is divided into a configuration part and a knowledge base.

The configuration part determines the configuration of the interpreters for the knowledge base. It consists of the metainterpreter, interpreters for different knowledge representation formalisms and the user interface. The knowledge base itself consists of different parts. They contain programs and data for the various interpreters of the configuration. For example, the rule part contains rule packages for the rule interpreter or the instruction part contains instructions for the metainterpreter.



BABYLON languages:

The following knowledge representation formalisms are available: objects, rules, Prolog and constraints. Each of them occupies a specific part in the BABYLON knowledge base. The global flow of control is defined separately in the instruction part. All formalisms allow to access the implementation language Common Lisp.



Objects:

The object part is often the basis for the other parts of the knowledge base: the rule, logic and constraint parts. Nevertheless, the frame interpreter is not superior to the rule, Prolog, or constraint interpreter. The object-oriented form of knowledge representation in BABYLON adopts the following main characteristics of the flavor system, a widespread object-oriented extension of various Lisp dialects. Objects are instances or occurrences of object types which are called frames in BABYLON (and flavors in the flavor system). A frame defines the attributes (or slots) of its instances (in the flavor system, these are the instance variables) and determines which behaviors are invoked by which messages to its instances. Such a behavior describes the actions which are performed when an instance receives a corresponding message (in the flavor system, these are the (primary) methods).

Frames can be organized in an inheritance graph and then contain as components frames whose characteristics they inherit. The direct predecessors are referred to as superframes or superior frames. Since a frame may possess several superframes, we speak of multiple inheritance. The algorithm according to which attributes and behaviors are inherited which occur in more than one superframe corresponds to that of the flavor system, i.e. mainly depth first and from left to right. Inherited behaviors can be modified by :after or :before encapsulations. Other forms of method combinations of the flavor system are not supported.

In the following aspects, the frame concept of BABYLON goes beyond the flavor concept:

Attributes can be connected with metainformation which can be evaluated automatically if required. This metainformation is stored in annotations. The user is able to define his/her own annotations in addition to those provided by the system. Furthermore, BABYLON admits active values as attribute values. They can be used as demons monitoring attribute values.

Rules:

Knowledge representation by means of production rules is one of the oldest formalisms used in the construction of expert systems. In the definition of the rule-oriented formalism in BABYLON, maximum power was not of primary importance, but rather a useful delimitation of characteristics with respect to other formalisms. In the rule formalism, the rules can be divided into packages which can be evaluated separately. The evaluation strategy of a rule-package is not defined in the package, but it is defined by the caller of the package. Forward and backward chaining with different control strategies are supported. Rules contain references both to constructs of other formalisms and to constructs realized in the underlying programming language. References to such constructs are forwarded to the metainterpreter for evaluation. The rule interpreter itself has no language of its own to represent facts and operands.

Prolog:

BABYLON provides a specific Prolog version for logic-oriented knowledge representation. Prolog is a programming language representing knowledge in form of Horn clauses. A Horn clause consists of an atomic formula, the conclusion, and an arbitrary number of further formulas, the premises. It denotes an implication from the premises to the conclusion. The Horn clauses are processed according to a fixed strategy (depth-first search with backward chaining) to prove hypotheses, i.e. arbitrary atomic formulas. The reference manual for standard Prolog is (Clocksin & Mellish 1984).

The special feature of BABYLON Prolog is its integration into the overall system. Prolog hypotheses can be used in other formalisms as conditions or for data inquiry. Conversely, BABYLON Prolog can use constructs from other formalisms, such as Lisp expressions or attribute and behavior references as premises. In particular, there are metapredicates defined on the constructs of other formalisms allowing to draw metaconclusions. As a side-effect, the number of system predicates can be reduced if compared with other Prolog implementations.

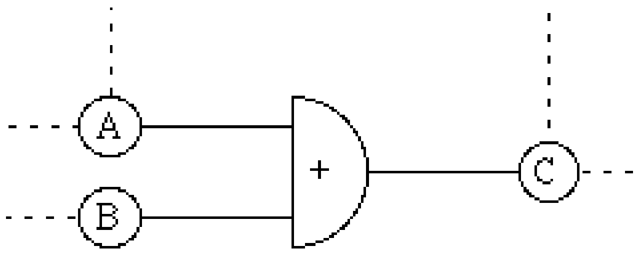
For modularization, the clauses in BABYLON Prolog can be combined to clause sets. The Prolog part of the knowledge base forms the first clause set, further clause sets can be located in separate files. The current clause sets are used for proving hypotheses.

BABYLON Prolog uses a Lisp-oriented syntax instead of the standard Prolog syntax.

Constraints:

Constraints can be used to model marginal conditions or constraints, such as physical laws or logical contexts. In metaphoric terms, we can regard the constraints as nodes of a network which interconnect variables thus establishing a connection between the variables. The following figure shows such a constraint connecting three variables A, B and C such that

A+B is equal to C.



The following table demonstrates the various effects of this constraint:

start values			values filtered by the constraint		
A	B	C	A	B	C
3	4	-	3	4	7
4, 5	3, 4	-	4, 5	3, 4	7, 8, 9
1, 2	3, 4	6, 7	2	4	6
3, 4	5, 6	1, 2	ø	ø	ø

Formally, a constraint consists of a set of variables and a relation on these variables. Using common variables between different constraints, we can compose constraint networks. If we predefine values or sets of possible values for a subset of the variables, a constraint network can be used as follows:

to check the consistency of values and

to compute values for unknown variables;

to filter sets of possible values, i.e. to eliminate inconsistent values.

The constraint interpreter of BABYLON is based on CONSAT, a domain-independent constraint system. The constraint language combines the simplicity of extensional constraint descriptions with the power of intensional descriptions. Hierarchies of constraints and (recursive) constraint networks can be constructed. Various control strategies are provided: apart from local propagation, there is a method combining this strategy with backtracking.

Lisp:

Common Lisp is the implementation language of BABYLON, but it can also be used for knowledge representation. In a knowledge base, Lisp can be used between all BABYLON expressions. Many BABYLON expressions admit or require Lisp at specific syntactic positions. The reference manual for Common Lisp is (Steele 1984).

To keep the knowledge bases portable, one should confine oneself to a subset

of Common Lisp. There is an abstract interface to the flavor system and a portable inhouse development of flavors.

Instructions:

Instructions define the global flow of control through the expert system. They are Lisp expressions which typically handle rule packages according to a specific strategy, make Prolog inquiries, activate behaviors or functions directly programmed in Lisp (e.g. input/output).

Integration of formalisms:

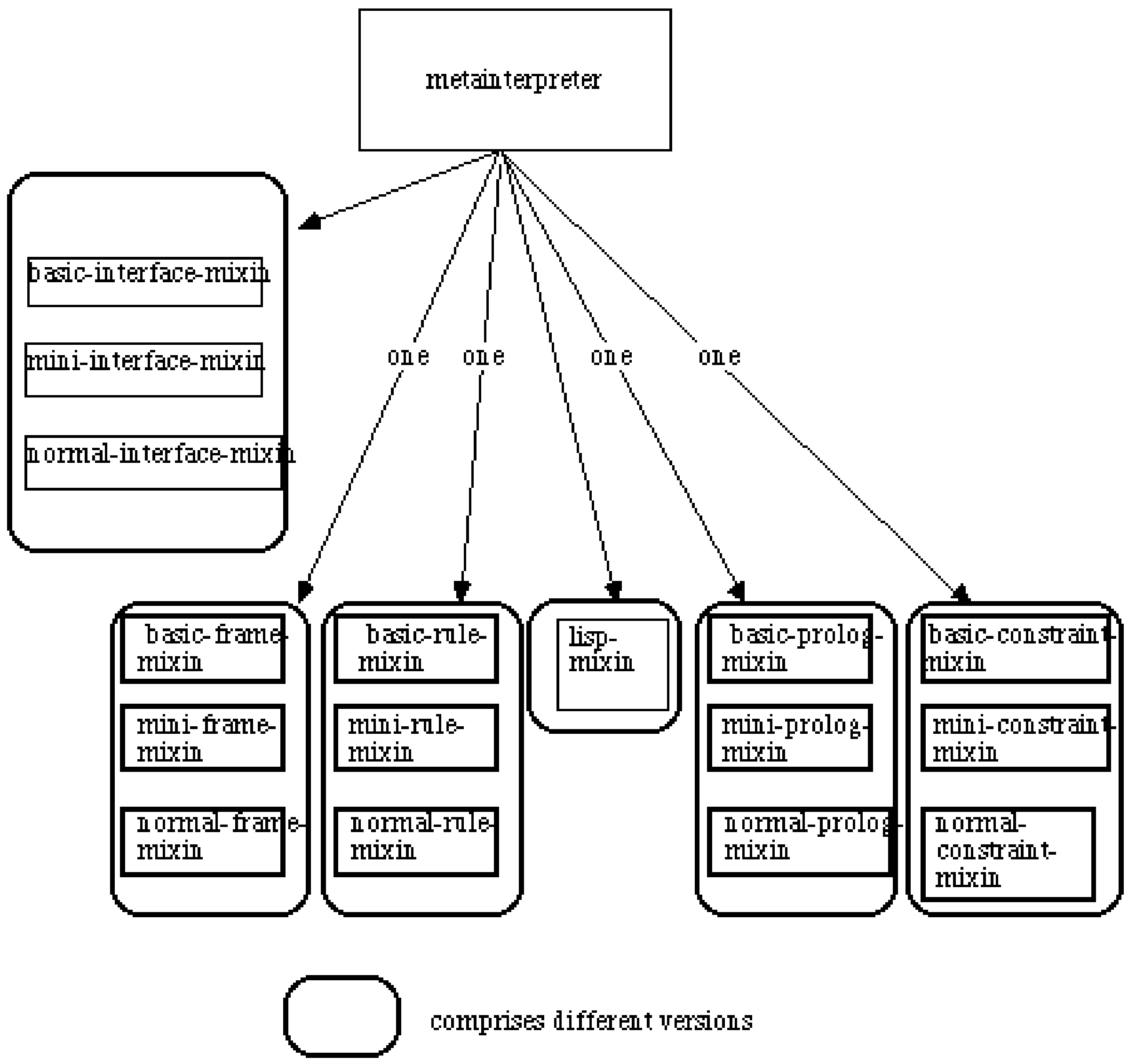
The following table shows the integration of the knowledge representation languages into BABYLON. A formalism X is usable in another formalism Y as described in column X and line Y:

	Lisp	objects	rules
Lisp	—	via messages to objects, and functions for objects	evaluation of rule packages
objects	as attribute values, in behaviors	---	not possible
rules	as conditions, as actions	attribute and behavior references as conditions and action	—
Prolog	Lisp-expression as premisses	attribute and behavior references, metapredicates	metapredicates
constraints	Lisp-expressions as value spezifications, activation conditions, in patterns	object references value specifications, activation conditions, in patterns	not possible

	Prolog	constraints
Lisp	proving hypotheses	defining and satisfying constraints
objects	not possible	via mesages
rules	in conditions and actions	satisfied-p expressions as premisses
Prolog	---	satisfied-p expressions as premisses or in candidate
constraints	Prolog expressions as activation conditions	---

BABYLON configurations:

Since all parts of a BABYLON knowledge base are optional, the knowledge base interpreter needs not always comprise all basic interpreters (specialists). Therefore, the interpreters for BABYLON knowledge bases are configurable. In addition to the metainterpreter which is absolutely necessary as a manager in each configuration, a configuration may consist of interpreters for Lisp, objects, rules, Prolog or constraints. Custom interpreters are also possible. Another component is the user interface. The last four interpreters and the user interface are available in three versions of different comfort.



User interface versions:

Basic user interface with a TTY-oriented interface.

Mini user interface extended by a command loop and command menus.

Normal user interface with more comfortable command menus. This is the interface to be used by the knowledge engineer normally. Though this user interface is within the standard delivery, its realization is machine-dependent.

Frame interpreter versions:

Basic version with frames, behaviors, instances, inheritance and annotations.

Mini version with additional possible values specification.

Normal version with additional active values.

Rule interpreter versions:

Basic version with forward and backward evaluation, various junctors and action types.

Mini version with an additional protocol component.

Normal version with an additional explanation component .

Prolog interpreter versions:

Basic version with clauses, clause sets and system predicates.

Mini version with an additional protocol component.

Normal version with an additional explanation component.

Constraint interpreter versions:

Basic version with constraints, constraint networks and various evaluation algorithms.

Mini version with an additional protocol component.

Normal version with an additional connection to the frame formalism.

System size:

```
   frame interpreter:              123 KByte
   rule interpreter:        128 KByte
   Prolog interpreter:             168 KByte
   constraint interpreter:         121 KByte
   total core system:              650 KByte
for machine-specific user interfaces:
   Macintosh:                       47 KByte
   Lisp machines:                   73 KByte
```

References:

Christaller, Th., Di Primio, F., Voß, A.(eds.), Die KI-Werkbank BABYLON, Addison-Wesley, Bonn 1989

Christaller, Th., Di Primio, F., Voß, A.(eds.), The AI Workbench BABYLON, Academic Press 1992

Clocksin, W.F.; Mellish, C.S.: *Programming in Prolog.* 2. edition, Berlin: Springer, 1984.

Fidelak, M.; Höffgen, K.U.; Voß, H.: *Spezifikation der K3-Mechanismen.* GMD-TEX-I-Bericht, GMD, Sankt Augustin, April 1987.

Früchtenicht, H.W.; Güsgen, H.W.; Hrycej, T.; Mörler, G.; Struss, P. (eds.): *Technische Expertensysteme: Wissensrepräsentation und Schlußfolgerungsverfahren.* Oldenbourg, München, 1988.

Gaines, B.R., Linster, M.: Integrating a Knowledge Acquisition Tool, an Expert System Shell and a Hypermedia System. *International Journal of Expert Systems* **3** (2), 1990, 105 129.

Güsgen, H.W.: *Constraints, eine Wissensrepräsentationsform -- Überblick.* Arbeitspapiere der GMD 173, Sankt Augustin, 1985.

Güsgen, H.W.; Junker, U.; Voß, A.: Constraints in a Hybrid Knowledge Representation System. In: *Proceedings of the IJCAI-87,* Milan, Italy, 1987, 30-33.

Güsgen, H.W.: CONSAT: Foundations of a System for Constraint Satisfaction. In: H.W. Früchtenicht et al. (eds.), *Technische Expertensysteme: Wissensrepräsentation und Schluß-folgerungsverfahren,* 415-440.

Müller, B.S.: *Lehrmaterialien BABYLON: Die Beispielswissensbasen zur Pilzbestimmung.* Arbeitspapiere der GMD Nr. 221, GMD, Sankt Augustin, September 1986.

di Primio, F.; Wittur,K.: BABYLON: A meta-interpretation-model for handling mixed knowledge representations. In: *Proceedings of the seventh International Workshop on Expert*

Systems and their Applications, Avignon, 1987, 821-833.
Steele, G.L., jr.: *COMMON LISP: The Language*, Digital Press, 1984.